Task

Train a minimal-size language model on Wikipedia that can generate coherent responses. Justify your choices in architecture, tokenization, and training strategy.

Data Collection

The data used for training was sourced from the Wikimedia Wikipedia dataset with the dataset identifier: “wikimedia/wikipedia”, version “20231101.en”. The dataset was loaded using the Hugging Face `datasets` library.

I chose a subset of **200,000 articles** from the dataset, randomly sampled to ensure diversity across various domains.

Data Preprocessing

Objective

The purpose of preprocessing was to clean, normalize, and structure the text data to prepare it for training the model effectively.

Steps Involved:

1. **Text Normalization:**
    - All text was normalized to ASCII using the `unidecode` library to convert accented or special characters to their closest ASCII representation.
2. **Article Splitting:**
    - Each article was split into sections based on double newlines (“”).
    - Sections were then further split into headings and paragraphs wherever a newline (“”) occurred.
3. **Exclusion of Unnecessary Headings:**
    - Unnecessary headings such as “Discography”, “Awards”, “References”, “External links”, “See also”, etc., were excluded from processing.
    - This exclusion was done to ensure that only relevant content was considered.
4. **Handling Long Paragraphs:**
    - Content with fewer than 80 characters was discarded to ensure only meaningful information was retained.
5. **Data Storage:**
    - Processed data was saved into arrays such as `final_headings`, `content`, `article_no`, `headings`, `paragraphs`, and `article_numbers` using Python’s `pickle` library.
    - Additionally, the data was saved in JSON format for future use.

Result

The preprocessing step resulted in a clean, structured dataset of articles that was ready for tokenizer training. The dataset contained **200,000 articles**, processed into paragraphs of at least 80 characters each.

Example Tokenization Output

Below are a few samples from the dataset to demonstrate how the text is structured and prepared for training.

Few Examples from the dataset:

- **Heading:** No Heading
  Content: HMP Hull is a Category B men's local prison located in Kingston upon Hull in England. The term 'local' means that this prison holds people on remand to the local courts. The prison is operated by His Majesty's Prison Service.
- **Heading:** History
  Content: Hull Prison opened in 1870, and is of a typical Victorian design. Ethel Major was the last person and only woman to be executed at Hull in 1934. She had been convicted of the murder of her husband.
- **Heading:** The prison today
  Content: Hull is a local prison holding remand, sentenced, and convicted males. Prisoners are employed in the workshops, kitchens, gardens, and waste management departments.

Tokenizer Training

Choice of Tokenizer

I chose the **Byte Pair Encoding (BPE)** tokenizer for this task due to its efficiency in handling a large vocabulary size while maintaining subword-level representations. BPE is particularly effective when dealing with a varied dataset such as Wikipedia, where words can be highly diverse.

Training Process

1. **Vocabulary Size:** 20,000
2. **Model Type:** BPE
3. **Sequence Length:** 128 tokens
4. **Tokenizer Framework:** `keras_hub` library was used for training the tokenizer.
5. **Special Tokens:**
   - `<pad>` : Padding token, used to fill up sequences to a fixed length.
   - `<unk>` : Unknown token, used to represent words not present in the vocabulary.
   - `<s>` : Beginning of Sentence token (BOS).
   - `</s>` : End of Sentence token (EOS).

Tokenizer Output

- The trained tokenizer was saved as a `.proto` file.
- Vocabulary was built with 20,000 tokens.

Vocabulary Size Justification

The vocabulary size of 20,000 was chosen after experimenting with smaller sizes of 5,000 and 10,000. Given the diverse nature of Wikipedia articles, a larger vocabulary provides better coverage of the unique words present in the dataset. A smaller vocabulary resulted in excessive fragmentation of words into subwords, leading to poorer coherence in generated text.

Increasing the vocabulary size beyond 20,000 was avoided due to computational constraints, as a larger vocabulary would significantly increase model size and training time. The model size increases linearly with vocabulary size due to the embedding layer and output layer dimensions.

Data Preparation for Model Training

Data Splitting

The dataset was split into training and validation sets using a **90:10 ratio**.
- **Training Data Size:** 529,108 samples
- **Validation Data Size:** 58,790 samples

File Storage

- Training data was saved to a file named `train.txt`.
- Validation data was saved to a file named `valid.txt`.

Dataset Creation

The training and validation datasets were created using the `tf.data` API, with the following considerations:
- **Batch Size:** 64
- **Minimum String Length:** 80 characters

The datasets were later saved using `tf.data.experimental.save()` for reusability.

Model Architecture

Model Choice

A **GPT-style model** was chosen with a **Causal Language Modeling (CLM)** training strategy, where the model is trained to predict the next token in a sequence given all the previous tokens.

Model Design

- **Token and Position Embedding Layer:**
  - The `keras_hub.layers.TokenAndPositionEmbedding` layer is used to combine the embeddings for the token and its position. This is crucial because the model must learn both the identity of the word and its positional context.
  - Masking is enabled by setting `mask_zero=True` to allow the model to ignore padding tokens during training.
- **Transformer Decoders:**
  - The model consists of 4 Transformer Decoder layers, implemented using `keras_hub.layers.TransformerDecoder`.
  - Each layer has **8 attention heads** and a **feed-forward dimension of 1024**.
  - The decoder layers use **default causal masking**, ensuring that the model cannot attend to future tokens during training.
  - Causal masking is necessary for autoregressive training, where the model generates text by predicting the next token one at a time.
  - The decoder layers do not use cross-attention since this is a purely generative model, not a sequence-to-sequence architecture.
- **Output Layer:**
  - A dense layer with a size of 20,000 (vocabulary size) is applied to the output of the final decoder layer.

- **No softmax activation** is applied to the output, allowing for the use of `SparseCategoricalCrossentropy` with `from_logits=True`.

Loss Function

- **SparseCategoricalCrossentropy:**
  - Used with `from_logits=True` since the final layer does not apply softmax.
  - This approach allows for numerical stability and computational efficiency when training the model.
  - Since we use padding tokens (`<pad>`), the model must be made aware of which parts of the sequence are valid inputs using masking.

Mask Handling

- Masking is applied using `keras.layers.Masking(mask_value=0)` to ensure that padding tokens are ignored during training and evaluation.

Evaluation Metric

- **Perplexity:**
  - Perplexity is a standard metric for evaluating language models, measuring how well a model predicts a sample.
  - It is calculated as the exponentiation of the cross-entropy loss, making it directly dependent on the model's vocabulary.
  - A lower perplexity score indicates a better predictive performance.
  - Perplexity of approximately **66.09** was achieved on the validation dataset.
  - Perplexity is influenced by the vocabulary size; larger vocabularies typically result in higher perplexity scores due to the increased difficulty in making accurate predictions.

Model Size and Comparison

- **Number of Parameters:** Approximately **13,451,808**
  - Embedding Layer: **5.15 million**
  - Decoder Layers (x4): **3.16 million**
  - Dense Layer: **5.14 million**

Training Process

Training Configuration

- **Epochs:** 5
- **Batch Size:** 64 (adjusted for multi-GPU training)
- **Callbacks:**
  - `ModelCheckpoint` for saving the best-performing model.
  - `TopKTextGenerator` for generating text samples at the end of each epoch.

Evaluation

Evaluation Metric

- **Perplexity:** Achieved a perplexity score of approximately **66.09** on the validation dataset.

Since this is a generative model trained with unsupervised learning, BLEU and ROUGE are not ideal metrics unless we have predefined references to compare against.
- **BERTScore:**
Measures semantic similarity between consecutive sentences, useful for detecting minor rewordings or paraphrasing and identifying abrupt topic shifts.
- **Sentence Transformer Similarity:**
Captures overall semantic consistency across sentences, useful for detecting coherence at a more global level by averaging sentence similarities.

Text Generation

- Used **Top-P Sampling (p=0.9)** and **Top-K Sampling (k=10)** for text generation.

Few Samples from the generated text:

Generated Text 1:
“In August 2015, he was appointed to the position of the new president in a position. In January 2016, he joined the new board of directors and vice president and was the president of the board of the new board. of staff in the board, a board member was appointed to the chairman of the board in August. members of the board voted to have two members, with the president, and two other members, and the chairman of the board are elected to the President.man, who has been elected to the board of directors, was elected a member of the board.”

BERTScore F1: 0.662
Sentence Similarity: 0.566

Conclusion

In this project, I successfully implemented a minimal-size language model from scratch, leveraging a large Wikipedia dataset for training. Through careful tokenization and model architecture design, the model achieved a respectable perplexity score, which indicates its ability to understand and predict text patterns effectively. I employed various sampling techniques, such as Top-P and Top-K, to generate coherent and contextually relevant text, demonstrating the model’s potential for real-world applications.

The architecture and training pipeline designed can be further improved by fine-tuning on more relevant datasets and increasing model complexity.

1. **Training and Evaluation Notebook (`training-redblock-gpt.ipynb`):**

- The model uses **Keras 3.0** and **Keras-Hub** for building and training a transformer-based language model.
- A **distributed training strategy (`tf.distribute.MirroredStrategy()`)** is employed for **multi-GPU training**.
- The **model architecture** includes:

    - **Embedding Layer**: `TokenAndPositionEmbedding` with a vocabulary size of **20,000** and embedding dimension of **256**.
    - **Transformer Decoder Layers**: 4 decoder layers with:
        - **8 attention heads** each.
        - **Feed-forward dimension** of **1024**.
    - **Output Layer**: Dense layer with **20,000 units** (vocabulary size).
- **Hyperparameters**:
    - **Batch Size**: 64 (increased by `strategy.num_replicas_in_sync` for multi-GPU training).
    - **Epochs**: 5.
    - **Learning Rate**: 0.0001.
    - **Vocabulary Size**: 20,000 (increased from 5,000 and 10,000 based on performance trials).
- **Loss Function**: `SparseCategoricalCrossentropy(from_logits=True)` .
- **Metrics**: `Perplexity` .
- **Optimizer**: `Adam` with gradient clipping (`clipnorm=1.0`).
- **Text Generation**: Used **Top-P Sampling (p=0.9)** and **Top-K Sampling (k=10)** for text generation.
- **Text Generation Evaluation**: BERTScore and Sentence Transformer Similarity

2. **Preprocessing Notebook (`Preprocessing_Wikipedia_Dataset.ipynb`):**

- **Data Collection**:
    - Dataset: **Wikimedia Wikipedia** (version `20231101.en`).
    - **200,000 articles** sampled for training.
- **Data Preprocessing**:
    - **Text Normalization** using the `unidecode` library to convert special characters to ASCII.
    - **Splitting Articles**: Based on double newlines and further split into paragraphs.
    - **Exclusion of Unnecessary Headings** like “Discography”, “References”, etc.
    - **Content Filtering**: Discarded paragraphs with fewer than **80 characters**.
    - **Storage Format**: Saved as `.proto` files using Python’s `pickle` library.

3. **Training Time & Hardware:**

- **Hardware Used**: 2 Kaggle GPUs (T4).
- **Training Time**: **5 hours** for 5 epochs.